Aalto University
School of Science
Degree Programme in Computer Science and Engineering

Jussi Päivinen

Development of route finding algorithm for dynamic ride sharing application

Master's Thesis
Espoo, October 5, 2016

Supervisors: Professor Lauri Malmi, Aalto University
Advisor: Miika Nordström M.Sc. (Tech.)

Aalto University
School of Science
Degree Programme in Computer Science and Engineering

ABSTRACT OF
MASTER'S THESIS

Author: Jussi Päivinen

Title:
Development of route finding algorithm for dynamic ride sharing application

Date: October 5, 2016 **Pages:** 54

Major: Software Technology **Code:** T-106

Supervisors: Professor Lauri Malmi

Advisor: Miika Nordström M.Sc. (Tech.)

Vedia Taxi is a mobile application that makes it possible to share taxi rides with people going to the same direction. It uses an algorithm to combine the routes of different people to achieve this. However the algorithm can only handle cases where people leave from the same location and people cannot join from the route. The purpose of this work is to extend the algorithm to handle these cases.

For the extended algorithm four routing services were compared. These routing services provide the raw route data and time estimates for the routes. Google Maps was chosen among these providers, because it had the most accurate time estimates for the routes. The raw routes from different providers had some differences, but all of them were adequate. While the Google Maps did not have the best route calculation time, it was crucial for the algorithm that the time estimates are as correct as they can be.

The extended algorithm was tested using test searches that mimicked real life taxi rides, as there was not enough test data from real life rides made with the application. The performance of the algorithm was tested by using the same test searches.

The test searches showcased that with the extended algorithm routes can be formed also when start locations are different. This also makes it possible to join along the route. The performance of the algorithm was highly dependant on the amount of time it took to calculate the raw routes. The calculation of routes took 92% and 98% of the running time of the algorithm. The average time to calculate raw route was 275ms and the evaluation of a single ride took 275-300ms.

Keywords: sharing economy, taxi, routing

Language: English

Aalto-yliopisto
Perustieteiden korkeakoulu
Tietotekniikan koulutusohjelma

DIPLOMITYÖN
TIIVISTELMÄ

Tekijä:	Jussi Päivinen		
Työn nimi:	Dynaamisen kyydinjakosovelluksen reitinhakualgoritmin kehittäminen		
Päiväys:	5. lokakuuta 2016	**Sivumäärä:**	54
Pääaine:	Ohjelmistotekniikka	**Koodi:**	T-106
Valvojat:	Professori Lauri Malmi		
Ohjaaja:	Diplomi-insinööri Miika Nordström		

Vedia Taxi on mobiilisovellus, joka mahdollistaa taksikyytien jakamisen samaan suuntaan matkustavien ihmisten kesken. Jakamisen mahdollistaa algoritmi, joka yhdistelee eri ihmisten reittejä. Algoritmi pystyy kuitenkin yhdistämään vain reittejä, joissa kaikki ihmiset lähtevät samasta lähtöpisteestä. Ihmiset eivät myöskään voi liittyä kyytiin matkan varrelta. Työn tarkoituksena on jatkokehittää algoritmia niin, että kyytiin olisi mahdollista liittyä mistä tahansa.

Uutta algoritmia varten vertailtiin neljää reitityspalvelua keskenään. Nämä reitityspalvelut tuottavat reittiohjeet ja aika-arviot reiteille. Google Maps valittiin parhaaksi reitityspalveluksi, koska sen aika-arviot reiteille olivat tarkimmat. Reittiohjeissa oli joitakin eroja eri reitityspalveluiden välillä, mutta kaikki niistä olivat tarpeeksi hyviä. Google Maps ei muodostanut reittejä kaikista nopeimmin, mutta algoritmin kannalta oli välttämätöntä, että aika-arviot olisivat mahdollisimman tarkkoja.

Algoritmia testattiin käyttämällä kyytejä ja hakuja, jotka vastasivat todellisia taksikyytejä. Tämä lähestymistapa otettiin sen takia, että Vedia Taxi ei ollut tuottanut tarpeeksi oikeaa testimateriaalia. Algoritmin tehokkuutta mitattiin samoilla testikyydeillä ja -hauilla.

Testihaut näyttivät, että uusi algoritmi pystyy yhdistämään reittejä niin, että ihmiset lähtevät eri aloituspisteistä. Uusi algoritmi mahdollisti myös sen, että ihmiset pystyvät liittymään kyytiin matkan varrelta. Algoritmin tehokkuus oli suuresti riippuvainen siitä, kuinka pitkä aika kului reittiohjeiden laskemiseen. Reittiohjeiden laskeminen kulutti 92%-98% algoritmin kokonaissuoritusajasta. Reittiohjeiden haku kesti keskimäärin 275ms ja yhden kyydin arviointi hakua vastaan kesti 275-300ms.

Asiasanat:	jakamistalous, taksi, reititys
Kieli:	Englanti

Acknowledgements

I wish to thank Vediafi Oy for the opportunity to make this thesis.

Special thanks to Miika Nordström, who was not only the instructor of the thesis, but provided thoughtful feedback on how the algorithm should work and what would be best from the users point of view.

For supervising and giving excellent feedback, I want to thank professor Lauri Malmi. The process of writing the thesis would have been much more painful without the regular and informative meetings and feedback.

Espoo, October 5, 2016

Jussi Päivinen

Abbreviations and Acronyms

HOV	High Occupancy Vehicle
GPS	Global Positioning System
FTS	Flexible Transport Services
ELY Centre	The Centre for Economic Development, Transport and the Environment
HSL	Helsingin Seudun Liikenne
UI	User Interface
GUI	Graphical User Interface
HTTP	The Hypertext Transfer Protocol
SDK	Software Development Kit
LS algorithm	Label Setting algorithm
LC algorithm	Label-Correctin algorithm
OSM	OpenStreetMap
OSRM	Open Source Routing Machine
Mbps	Megabits per second

Contents

Chapter 1

Introduction

The rise of the smart phones has made possible for everyone to have a portable computer in their pocket and the fast mobile networks make these computers connected. Together they form a platform with endless possibilities for applications that were not possible before. They also make old ideas easier to implement, like ride sharing.

The idea of sharing a car, taxi or a minibus is not new, but services that offered solutions for ride sharing have suffered from not being able to reach potential co-sharers in real time, making it difficult to organize shared rides that are not regular trips or long distance routes that have been planned beforehand. The reasons for wanting to rideshare are usually the lower price of the ride, lower emissions or social reasons. It might also not be possible to get somewhere without a car and especially in urban areas not everyone has their own car. Taxis are a possibility in these situations, but in Finland they are deemed to be bit too expensive and public transport is not flexible enough in all of the situations. That is why services like Uber[1] and KutsuPlus[2] have been great success with the customers. They have proved that dynamic ride sharing services interest people and that the new technology like improved personal positioning services and ubiquitous mobile payment solutions have helped them greatly to achieve ease of use that has not been offered before.

Vediafi has developed Vedia Taxi application to make a new kind of solution that makes it easier to share taxi rides between people that are going to same direction by combining the routes of different passengers to one shared route. There are similar services in other countries like Share-a-taxi [3] but Vedia Taxi is designed spesifically for the Finnish taxi service.

1.1 Problem statement

The Vedia Taxi application has a fairly trivial route finding algorithm that assumes that all the passengers leave from the same spot. This makes it impossible to find routes that start from a different part of the area you are in, but the route would go near you. It also does not take in to account any disruptions in the route or traffic data when it calculates the combined route. The purpose of this thesis is to improve the route finding algorithm to make it possible to join the ride from different locations. We also want to take distruptions such as traffic incidents or traffic jams in to account when calculating the final route. We must also consider which routing service to use, as the time we spend calculating the routes can be considerable.

Research questions:

- How to find a route for people that start from different locations and go to different locations?
- How to get the routes needed by the algorithm, with local- or network-service?
- How to incorporate disruption data in to route calculation?

1.2 Structure of the Thesis

Thesis starts by introducing the basic concepts of sharing economy and spesifically how taxis could be shared. Next we explain the current state of flexible transport services in Finland and various ridesharing services that exist or have been tried. After that the thesis goes on to the details of the current implementation of Vedia Taxi by going through use cases and presenting the technical background.

Next we go through the details of finding the shortest route in a road network and compare the potential routing services that the extended algorithm could use. After that we talk about the implementation details of the algorithm and present the new use cases and problems brought by the solution. Finally we have an evaluation of the features of the extended algorithm and its performance. In the last chapter we have the conclusions and discussion about the success of the thesis.

Chapter 2

Background

This chapter first presents the concept of sharing economy and its applications with cars and taxis. Then we go through the state of flexible transport services in Finland and compare them in the categories of price, flexibility and availability.

2.1 Sharing economy

The concept of sharing economy emerged during the global financial crises in the past ten years. The need for alternative measures to acquire digital or physical goods as well as services rose because getting these commodities with money was not preferable anymore.[4] There has also been call to more sustainable methods of consumerism as environmental consciousness continues to grow.[5] Sharing economy can be defined as a host of systems that enable people to share goods, services, data or talent. These systems can be economic or social systems and they usually take advantange of new technology. New technologies enable the systems to reach more people and make it easier for people to share their expertise or unused items and make use of these shared goods themselves.[4]

Today there are numerous examples of these kind of services in all areas of business. Airbnb challenges the hotel business by allowing people the rent out their vacant summer cottages or even their own houses while they are away.[5] Aalto University has their own marketplace, where you can offer and request services or lend items.[6] Uber has allowed people to easily offer ride services with their own cars when they have the time.[5] Services like Wikipedia or GitHub represent different kind of sharing. They share information and programs that have been generated by other users and make it possible for everyone with an internet connection to use this information.

Although financial savings and sustainability are big motivations to support sharing economy, the main motivation to actually participate in it is enjoyment. People want to share things from the pure pleasure of helping others and the way it provides a meaningful way to belong to a community. [7] And statistics show that in 2013 in UK 64% of adults have participated in sharing economy and 52% of American adults have rented or leased items they would usually buy. Additionally 83% of the Americans said that they would participate if it was easy to do. [8]

In the following sections we are going to focus on the means of sharing transportation.

2.1.1 Carpooling

Carpooling can be defined as sharing a private vehicle regurarly with other passengers to commute or go to school. In an ideal case, it offers the flexibility of single occupancy vehicle use without the costs of maintaining and owning your own car. The ideal case is usually constrained by the combining of various schedules and locations when organizing the actual ride. [9]

Carpooling emerged in North America during the Second World War as a mean to conserve resources for the war and can be seen as the first form of ride sharing. Goverment enforced carpooling with laws and workplaces were required to have bulleting boards where carpools could be formed. Later in the 1960s to 1980 use of carpooling rose again in response to the energy crises. Again there was help from goverment and workplaces to motivate workers to participate in carpooling such as high occupancy vehicle (HOV) lanes, priority parking privileges and ridematching programs. In the present day carpooling has transferred in to the Internet along other services, but there is not that much pressure or support from the goverment anymore to use carpooling, so the popularity has not changed much from the 1990s. [10]

The main reason for using carpooling in the present day is the use of HOV lanes followed by the enjoyment of the company of other people while traveling. Almost as important were travel time savings, sharing costs and helping the environment. [11]

2.1.2 Dynamic ride sharing

While carpooling is a contract for recurring trips with usually the same people, dynamic ride sharing aims to provide rides with short notice and with different participants. The characteristics of a dynamic ride sharing service usually include the possiblity to offer and request rides on real time and automatically match the rides between different parties. While dynamic, the

rides are still prearranged, so they differ from hitch-hiking or casual ride sharing in that manner. Dynamic ride sharing enables people to cut their travel costs and offer better mobility, much like carpooling. [12]

Earliest forms of dynamic ride sharing worked through telephone services that provided information on potentially matching ride on the parameters provided by the caller. Nowadays the services use mobile phone applications to leverage the GPS services and internet connection for easier and faster matching. [10]

Mobile phones also enable paying for the driver without cash or physical credit card which not only makes the payment easier, but the ride safer for all participants. Ride safety can be an issue with dynamic ride sharing, as the driver and participants do not know each other. This consern can be alleviated by requiring all parties to provide some identification information about themselves through social media authentication or other kind of registration to the service. The service can also track the participants and ensure that the ride was completed succesfully. Having an review system for both the driver and the passengers helps to build a trust for future matches between the participants. [1]

2.1.3 Sharing taxis

In dynamic ride sharing private vehicles were used, but it is also possible to share rides with existing services like taxi service. Taxi service is one of the more flexible and easy to use services, but usually they are used privately. [13] Sharing taxis can provide more profits to the taxi driver by having more customers and cheaper rides for the passengers. It can also reduce the waiting time for a taxi in a congested situation, as the number of taxis available is usually limited. [14]

Taxi sharing service can be implemented much like dynamic ride sharing service and it can be integrated with taxi services to enable payment and ordering of taxi with mobile phone to make the process easier. Example of this kind of system is SHARE-a-Taxi. [3]

2.2 State of flexible transport services in Finland

Flexible transport services (FTS) are services offered for private customers that have flexible route, vehicle allocation, vehicle operators and type of payment. Each of these variables can vary from being set a long time before

actual ride to being set moments before the ride is commencing, the latter being the more flexible type. The range can be seen from the traditional fixed route, fixed timetable public transport to hired taxi services that are available around the clock. [15]

In this section we are going to have a look at and compare the FTS available for private customers in Finland, as Finland is the focus of Vedia Taxi service as well. We are leaving the normal public transportation services like buses, trains, trams and subway out of this comparison, because they are on fixed routes and timetables and cannot be used for real flexible transportation.

2.2.1 Taxis

In Finland, the official taxis operate 24 hours a day, every day of the year. You can order a taxi to any address, including addresses in sparsely populated areas. Orders can be placed by calling, sending a text message or by mobile application Valopilkku. There is approximately one taxi per 530 people. Taxis operate with taxi permits that are controlled by ELY Centres. Different municipalities have their own call centers where you can order the taxi, but the maximum price is set by ELY Centres and it is the same throughout Finland.

The price of a taxi is calculated by taking a basic fee and adding a kilometer based fee. Basic fee is 5.9 euros during 6-20 on weekdays and 9.0 euros other times and kilometer based fee that starts at 1.55 euros per kilometer and increases when there are more passengers. There is also a waiting fee of 44.6 euros per hour if the taxi is moving especially slow in congested traffic or when passengers are making the taxi wait. [16]

2.2.2 Airport Taxi

Airport taxi is a private company that offers fixed price transports to the Helsinki-Vantaa airport from the Helsinki metropolitan area and the other way around. It also operates any time of the day or year. The prices are considerably smaller, if you are willing to share the taxi with people going from or leaving to the same areas. The company makes up the route and all passengers have their own fixed price based on the area they are going or leaving from.

Price for one passenger varies from 19.5 euros to 39.5 euros depending from the area. [17]

2.2.3 Uber

Uber is a private company that offers peer-to-peer ridesharing services. People can register as drivers and passengers can order a ride with mobile application that shows approximate price for the trip and the details of the driver once you have ordered the ride. In Finland Uber is currently available only in Helsinki area. Because it is a peer-to-peer service the availability depends on the amount of people currently offering rides. There is no guarantee of service.

Price of a ride consists of base price and kilometer based price that is dependent on the city Uber is operating in. In Helsinki the base fare is 2.0 euros and additional 0.2 euros for waited minute and 1.0 euros for driven kilometer is added to the price. There is also a special surge pricing when the demand is high, which can cause price to change with multiplier with no upper limit. [18]

Although the service itself is legal in Finland, the drivers who drive the passengers should have a license to drive normal taxi too, according to Ministry of Transportation and Communications. [19] Uber does not require drivers to have taxi license. [18] Finnish police has fined the drivers that do not have a taxi license if the police has encountered them while they are driving a passenger. [20]

2.2.4 Kutsuplus

KutsuPlus is a service developed by Helsingin Seudun Liikenne (HSL) that combines professional drivers driving minibusses and people sharing them to get from one bus stop to another. There are no fixed routes, the routes are made up dynamically when people request rides through the KutsuPlus mobile application. Passengers can join and leave at any bus stop in the Helsinki area. The service is online from 06 to 24 on weekdays.

The price of the ride has several components. There is a base fee of 3.5 euros and kilometer fee of 0.45 euros. But there is discount of 20 percent if you use the service between 10 and 14. If you have more than one passenger joining with the same order you get discounts up to 50 percent of the price if there is more than five people joining with the same order. [2]

The service is currently offline due to it being too pricy for HSL to keep up. There are plans to revive it during this year if a private company wants to take the service and run it or if HSL comes up with a new business model that will make it more cost-effective. [21]

2.2.5 Vedia Taxi

Vedia Taxi is a mobile application that makes it possible to share taxi rides with people going to the same direction, much like KutsuPlus. People leaving from the same area can search existing rides or make a new one and Vedia Taxi tries to combine the routes of different rides which will then be carried out with official Finnish taxis.

The price of the taxi ride is split among the passengers so that passengers only pay for the part of the kilometer fees of the ride they were in the taxi. The basic fee is split evenly among all passengers. Vedia Taxi takes provision from the passengers that is 1.0 euros per every starting 10.0 euros of the passengers' share of the ride price.

2.2.6 Comparison between the FTS services

In this chart there is a rough comparison between the services presented in this section. The services are not directly comparable because KutsuPlus is not in use at the moment and Uber is not strictly legal in cases where the driver does not have a taxi license. Also Airport Taxi, Uber and KutsuPlus are only available in the Helsinki metropolitan area and not in the other parts of Finland. There is no waiting times included in the prices. [16] [17] [18] [2]

Service	Flexibility	Availability	Price for 20 km ride on a weekday
Taxi	Ideal	00-24 every day	36.3 euros
Airport Taxi	Only to or from airport	00-24 every day	29.5 euros
Uber	Ideal	When there are drivers	22.0 euros
KutsuPlus	Almost ideal	06-24 on weekdays	12.5 euros
Vedia Taxi	Almost ideal	When there are fellow passengers	20.15 euros (with two passengers)

Chapter 3

Environment

In this chapter we go through multiple use cases of Vedia Taxi and through them present the different functionalities of the application and the problems that may present when using it. Rest of the chapter is about the technical implementation of the Vedia Taxi server and application and the business model of Vedia taxi.

3.1 Example use cases of Vedia Taxi

3.1.1 General succesful use case

In this example we have two persons, Alice and Bob. Alice has not used Vedia Taxi before and Bob has used it and configured the application to his personal preferences.

Alice needs to go to the airport leaving from her home, but the price of a taxi seems a bit steep. Alice installs Vedia Taxi and fills her credit card and payment account information to be able to use the application and then proceeds to search for available rides. Alice chooses a starting location that is near her house from the map and searches for end location by typing first few letters of the name of the airport and then choosing the airport from the results. The search for rides yields no results so Alice creates her own ride from the search parameters and waits for other passengers going to the airport. While Alice waits she also saves the location of her home to her favourite locations so next time it is easier for her to search for rides.

Bob has setup a ride watchdog in Vedia Taxi, which informs him with push notification if a ride is created that matches the parameters in his watchdog. Bob is a frequent flier and is going to the airport today and receives notification that there is ride available. He sees that Alice is leaving

at a convenient time and has an appropeiate rating so he joins the ride. Alice receives a push notification that Bob has joined the ride.

When the start time is closing Alice orders the taxi and finalizes the ride so the route of the ride and passengers are final. Bob receives push notification that it is time to pay his share of the ride. Bob inserts his payment pin code and pays for the ride using the credit card information he provided when setupping the application. Alice receives push notification that payment is complete. Bob arrives to the start location of the ride and calls Alice using phone number available in the passenger information. Alice and Bob meet and take the taxi to airport.

When the ride is over Alice pays for the taxi and both parties mark the ride complete and rate the other party using a 5-star rating system. Later Vediafi relays the share that Bob paid from the ride to Alice.

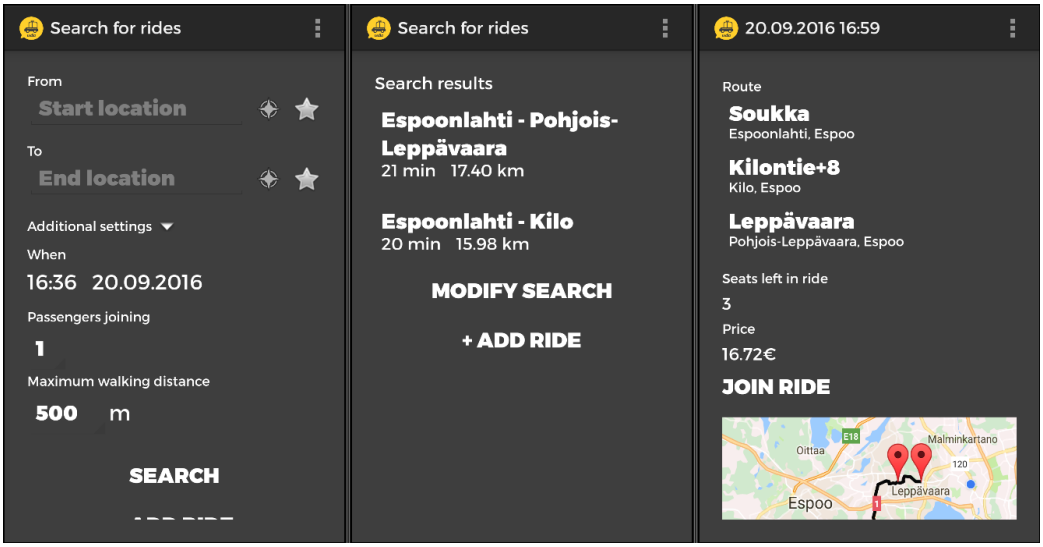

Figure 3.1: GUI of Vedia Taxi. On the left there is search screen, on middle search results and on the right the view where you can see details of the ride and decide if you want to join it

3.1.2 Use case with few problems

In this example we have three persons, Alice, Bob and Carol. All of them have used Vedia Taxi before.

Alice, Bob and Carol are all out in the town and it is late in the night and last buses have already gone. They all still need to get home and taxi is the only way to get there so Alice tries to search for rides in Vedia Taxi. When

she finds no rides, she makes her own ride and waits for other passengers. Both Bob and Carol need a ride too and they find Alice's ride in the search. Bob is going to the furthest location in the ride so Bob becomes the owner of the ride so he is going to be the one who pays for the ride and orders the taxi. Due to the high demand for taxis Bob can't get a taxi before the ride's start time hits. Bob then cancels the ride and Carol and Alice receive a push notification that the ride has been canceled.

Bob calls his friend for a ride because he cannot find a taxi, but Alice and Carol have no other choice but to try to get a taxi again. Alice makes a new ride and Carol joins it. Alice manages to get a taxi this time and marks the ride finalized and Carol tries to pay her share, but the payments fails because the credit card she has configured for Vedia Taxi has expired. She quickly corrects the information about the credit card and tries the payment again and it succeeds. They finish the ride normally and Alice pays for the taxi and they give ratings to each other.

3.1.3 Another use case with different problems

In this example we have three persons, Alice, Bob and Carol. All of them have used Vedia Taxi before.

Alice, Bob and Carol are all coming back from holidays and decide to take a taxi from the airport. Carol makes a ride after search did not yield any results and Bob and Carol join it. After all three are in the ride Alice takes a look at the route and it is not optimized in her mind. She reorders the stops so that Bob becomes the owner of the ride. Bob and Carol receive notifications about the change of the route.

They will take a taxi from the taxi stand and Carol and Alice try to pay their shares. Alice's card is declined because she spend all her money during her holiday and she does not have any other payment method, so she has to leave the ride. Because the route and the prices change Vedia Taxi informs the other passengers of this incident and they have to decide if they still want to continue with the ride. Both decide that it is still better to take the taxi with just two passengers and they continue normally with the new route. If one them should have rejected the new price or route, the ride would have been canceled.

3.2 Summary of potential problem situations

Here is a summary of problems that may arise when using the application. Some of them were already presented in the use cases. There can be both

technical and user related problems. For some user related problems there is not a clear solution.

- When user is filling in his payment information he does not have account number with him and that prevents him from using the application.
- User has either forgotten his pin number or his credit card does not accept payments so he cannot pay for the trip.
- One or more passengers do not show up to the starting location before the time the taxi is supposed to leave. In this situation the owner of the ride can kick the passenger from the ride and the rest of the passengers can decide if they want to continue with the ride anyway.
- Ride owner cannot get a taxi because of high demand or taxi cannot get to the start location in time due to traffic or other reasons. In this situation Vediafi refunds the payments if passengers have already paid to the owner of the ride.
- Taxi cannot keep up with the ride schedule because of traffic. In this situation the taxi ride can be stopped if passengers can arrange a better form of travel to themselves. Refunds for part of the ride must be negotiated with Vediafi separately.
- The ride cannot be formed because of unexpected bugs in the server or mobile code.
- The ride cannot be formed because of unexpected downtime in the third party software used in the server or mobile application.
- The ride cannot be completed because taxi breaks down or gets in to an accident. Here we can use the same approach as when taxi is stuck in traffic.

3.3 Technology stack

Vedia Taxi consist of a server and a mobile application. The purpose of the server is to store all the information about rides, users and payments and process all the requests made by the application. The server handles all the processing and calculations and the application serves as an UI for making the requests and showing results. The server and the application transfer information through HTTP-requests and push notifications.

The server is a virtual server running CentOS 7.2 operating system. [22] The server application is implemented with Django 1.8 [23] using PostgreSQL 9.4 [24] as the backing database. The programming language used is Python 2.7. The push notifications to mobile application are sent using Google Cloud Messaging API. [25] The current service used for getting raw directions data is MapQuest Directions API. [26]

CentOS was chosen as the operating system because it is free and very stable, because it does not use the latest feature versions of packages but ones that have been proven stable. [22] Django and PostgreSQL were chosen as the server development stack because of the previous experience development team had with them and the fact that they have had good performance, flexibility and stability in the past. Google Cloud Messaging has solid integration with Android and good examples in Python so it was natural choice for push notification service. [25] MapQuest Directions API was chosen because it had all necessary functionalities in its API and it had no request limit with free subscription. [26]

Mobile application is developed with Java 7 programming language targeting Android platforms upwards from 4.0.1 [27] using Android Studio for programming and designing the UI [28]. Application uses Facebook Login and Google Sign-in to authenticate and identify users. [29] [30] Currently the mobile application is available only for Android, but iOS-version is under development. Android was chosen as primary mobile platform because of the market share it has worldwide and in Finland. [31] [32] Facebook Login and Google Sign-In were integrated in the application because they make the user experience smoother.

Both the server- and mobile application use Stripe as payment solution. Mobile application uses Stripe SDK to create payment- and payment card requests. Server uses the Stripe API to actually make the payments and create new customer accounts to Stripe. Stripe was chosen as the payment platform because of the exceptionally good documentation and good examples in Python for the server and SDK for Android. [33]

3.4 Business model

You can install the Vedia Taxi application and search for the rides for free. For every completed ride Vediafi takes a provision from everyone that participated in the ride. Size of the provision depends on the size of the payment share. The provision is 1.0 euros per every starting 10.0 euros of the share price.

Chapter 4

Implementation background and methods

In this chapter we go through the theory behind finding the shortest routes in a road network and how road networks are presented for the algorithms. We also explain how disruption data can be acquired and used in these calculations. After that we compare the different map sources and direction data providers and go through the different aspects that might affect the development of the ride finding algorithm.

4.1 Background for finding shortest routes

Finding the shortest route in road network can be represented as finding the shortest path in a weighted directed graph, where road segments are the edges of the graph and nodes represent locations. Edge weight is the distance of a road segment. There are no negative weights in this graph, because distances of road segments are always positive. The search case we are looking closely is so called one-to-one search. In one-to-one search we want the shortest path from source node to one destination node only, not all of the nodes in the road network or some of them. [34]

4.1.1 Optimal shortest path algorithms

For optimal shortest path algorithms, there are two main approaches: label-setting (LS) algorithms and label-correcting (LC) algorithms. Label means the label of the node, which is the distance from the source node to this node. The difference between the approaches is the way they go through the nodes. In LS algorithms the label of a visited node can be only set once and then

it becomes unvisitable in contrary to LC algorithms that can make multiple visits to all the nodes and update the current estimation of a visited nodes label. In LS algorithms we can stop the search when we reach the destination node but in LC algorithms we must have final label for every node before we can be sure that the label of the destination node is final. This does not mean that LS algorithms are always better in one-to-one search, which one could intuitively assume. [34]

Dijkstra's algorithm is good example of LS algorithms. It visits every node once, until the destination node is found. First we mark the source node as visited and its label to zero, because the distance from source node to destination node so far is zero. Then we add all the nodes reachable from currently visited nodes to set of potential nodes for next visit. After that we take the node with smallest distance from source node and label it with that distance and add the nodes reachable from that node to the potential node set. If there would be duplicate nodes in the set, we just update the distance to that node with the smallest distance available. We continue this process until we visit the destination. The label of the destination node is the shortest distance from source node to the destination node. For the shortest path we can use a predecessor array in which we mark from which node we arrived to the visited node. Using this array we can calculate the shortest path backwards from the destination node to the source node. [35]

For the label-correcting algorithms, we present the Bellman-Ford algorithm. First we set each of the labels to positive infinity and the source node's label to zero. Then we go through each of the edges between the nodes. We update each of the edge's destination node, if the edge allows for shorter route than the node's previous label indicates. We do this same procedure N times, where N is the number of nodes in the graph. The final result has every node labeled. We can use simial predecessor array as in Dijkstra's algorithm to acquire the shortest path to any given destination node after tha algorithm has stopped. [35]

4.1.2 Heuristic shortest path algorithms

For road networks the optimal algorithms can be computationally intensive, as networks are big and there are numerous segments to be explored that can be in the exact opposite direction as the destination we want to get to. Heuristic algorithms address this problem by leveraging information about the relationship of source node and destination node and the structure of the road network in a larger scale. They use a heuristic function that approximates the likeliness of the new node to be on the shortest path in addition to the weight of the edge that leads to the node.

Good example of a heuristic algorithm is A* algorithm. It is very similar to the Dijkstra's algorithm, as Dijkstra can be seen as a special case of the A*. Similarly to the Dijkstra, A* is a LS algorithm. It goes through the nodes in the same way as Dijkstra's algorithm, but the labels of the nodes are calculated differently. The label consist of the distance from the source node to this node and an added heuristic value, which is the approximated distance to the destination node from this node. To be able to achieve optimal result using this method, the heuristic function must be admissible, meaning that it can never overestimate the approximated distance to the destination node.

As Dijkstra's algorithm can be seen as a special case of A* where the heuristic function always returns zero, the actual performance gain from using A* instead of Dijkstra depends solely on the quality of the heuristics function used. Fu et al. found that using A* instead of an optimal LS algorithm yielded 10-30% faster search times in transportation network graphs if appropiate heuristic function was used. [34]

4.1.3 Dynamic road network graphs

Algorithms presented before work well with graphs where edge weights are known beforehand and where the edge weights do not change while the algorithm is traversing the graph. But in road networks the edge values can be dynamic due to for example weather conditions, road work or traffic. In these graphs we cannot use the traditional shortest path algorithms to obtain the shortest path, but at best make a good assumption what the shortest path might be if the edge weights do not change while we traverse the graph.

For these graphs we must consider using so called policies to choose the best edge from each node in path from source to destination node. Policy is a set of rules that the graph traverser uses when it chooses which edge should be taken next. The information the traverser has increases during the traversal as we learn the true cost of the edges and the rules must take this into account. The cost of a policy is the same as the combined weights of the edges we traverse. Because the weights might change during the traversal, policies are rated by the smallest possible excepted cost and not actual cost. [36]

In his paper Polychronopoulos presents a dynamic programming method to find the optimal policy in a random graph where they have statistics on propable edge weight values. The real values are observed when we arrive to a node that is included in the edge. [36]

4.1.4 Disruption data sources

The traffic- and other disruption data needed by the dynamic road network algorithms needs to be gathered and refined before it can be used in calculations. With both America and Europe having their own models for forecasting weather globally, forecasts are easily available for everyone to use in these algorithms no matter where the road network resides. [37] [38]

Traffic flow data can be collected in three main ways. Oldest way of gathering data has to do with physical detectors on or beneath the road. These include detectors like pneumatic road tubes that detect car tyres passing over them and magnetic loops embedded in roads in square formation. The loops detect vehicles passing by monitoring the changes in the magnetic field generated by the loop. These methods have the downside of breaking due to weather conditions and sometimes heavy vehicles and the upside of having accurate and up-to-date information about the speed and the amount of traffic.

Second way to gather traffic information is having physical detectors on the side or above the road, that visually track the traffic. These devices include cameras, radars and infra-red detectors. Manual counting by trained obververs can be included as visual detector too. Visual methods mainly suffer from bad weather conditions and the difficulty of separating vehicles in high traffic. The upside is that these devices are rarely destroyed by the vehicles. [39]

Both the first and second method suffer from the need of installation and maintanence of physical devices on the roads so scaling these methods to cover wide areas is difficult. [40] The third method does not suffer from this problem, as it tries to leverage the current devices drivers have in their cars, such as mobile phones with GPS capabilities. [39] Google has successfully utilized this method with Google Maps, which shows live traffic data that has been collected anonymously from drivers that have Google Maps on for navigation or other purposes. Google Maps also makes use of traffic history that it has in predicting what the traffic might look like in the future hours. It can also tell you if the current traffic is normal or heavier than it usually is. [41] The downside of this method is that you have to have about 2-3% of the drivers reporting with GPS turned on and you have to make sure that there is no privacy issues collecting the location data. Upside is that GPS data has all the necessary information to make the traffic data calculations and there is no installation and maintanence costs of equipment involved. [40]

Getting data on traffic incidents can be done through two main methods. First we can analyze traffic data and compare it to historical data from the

same areas. This method naturally requires historical data of the area to be available. [42] Incidents can also be collected by crowdsourcing them from other drivers using mobile applications like Waze. The downside of this method is the fact that other drivers need to be using the same application that you are using in order for you to get the reports. [43]

4.2 Map sources

4.2.1 OpenStreetMap

OpenStreetMap (OSM) is a project founded in 2004 that aims to provide free map data that anyone can edit and access. Most of the map data is produced by volunteers around the world. It can be done by either tracing roads and other features from satellite imagery or by doing groundwork with GPS-enabled devices that allow tracking the position of features. Authorities from different countries have also donated official mapping data to OSM. The infrastructure that holds and serves the map data is also developed purely by volunteers.

Big challenge that comes with crowdsourcing is the validity and accuracy of the gathered map data. [44] After 2004, many reviews have been conducted in different countries. In 2008 in England and in 2009 in Germany studies compared the OSM data to a mapping data provided by authorities or commercial navigation systems and the results showed that OSM has very good mapping of the urban areas, but gradually weakens towards the rural areas. Study conducted in Germany in 2011 compared the OSM data on total road network of Germany to a commercial navigation systems data and the result was that OSM had more data, but lacked 9% of the data related to car navigation. The study also approximated that based on the history of the growth of OSM data, the 9% gap would disappear by the end of 2012. The result shows that at least in the countries that OSM has good coverage, it can be used for map applications.

Other problems include the fact that most of the data is produced by small percentage of the userbase, so some locations are more extensively mapped than others. OSM had 330000 members in 2010 of which 12000 or 3.5% provided 98% of the data. The percentage of active contributors has remained roughly the same throughout the project, but the total number is still rising and thus the number of active contributors is also rising. [45] The current number of registered users as of 2016 is 2.4 million. [46]

4.2.2 Commercial maps

There are many commercial propietary mapping solutions available, but we are going to focus on Google Maps, as it is the clear market leader [47] [48] and one of the few from which any reviews of their map data accuracy could be found. Google Maps is an online map developed by Google that was launched in 2005 and it covers the whole world. Google offers restricted editing of its maps. Mostly the editing is in the form of reporting incorrect features and adding new points of interests to the maps with their Map Maker tool. [49]

Comparing OSM data to Google Maps data directly is not possible because it is not possible to get the underlying data Google Maps uses to produce the maps. According to study conducted in Ireland 2010 that compared the two by spatial coverage, currency and ground-truth positional accuracy, there is no clear winner. Both had their own inconsistencies and errors. These results are for five smaller and larger cities and Ireland and can't be generalized for the whole world, but give some hint of what the differences are in urban areas for the two. [50]

4.2.3 Comparison

Name	Map source	Satellite maps	Price for web	Price for mobile
OpenStreetMap	Volunteers	No	Free	Free
Google Maps	Commercial	Yes	25000 daily requests free	Free

[51] [44]

4.3 Routing services

In this section we cover four possible routing solutions for our algorithm: MapQuest, Google Maps, Routino and Open Source Routing Machine (OSRM). Google Maps has their own map data and the other three use OSM as their mapping system. Routino and OSRM are offline solutions, meaning that you need to download OSM data from the area of your choice and produce routing data locally after the data has been prepared for the routing algorithm. The other two are online solutions which have an API that provides the routing data.

Biggest difference in the offline- and online solutions is the fact that online solutions have a chance to use real time traffic data and take traffic incidents and weather in to account when they decide which route is the best one to

take and how long it takes to reach your destination. Although MapQuest does support this feature in theory, it does not have sufficient data in Finland to produce meaningful results yet. Google Maps does support this feature in Finland.

If you use an online solution you don't have to maintain the map data yourself. If you maintain the data yourself you know how fresh the data is but it takes some extra maintaining effort to do so.

The biggest upside of offline solutions is that you can install them to the same server that Vedia Taxi resides and it eliminates the time needed to make the call to a network API and download the results. The calculations done by the routing algorithms need extra hardware for the server, which can increase the costs, but Google Maps and MapQuest both have strict usage limits in their free subscription mode, so using them is also going to produce some extra costs. [52] [26] [53] [46]

4.3.1 Comparison

The four routing services were compared using production level server described in section 3.3 and a stable 1000/1000 Mbps internet connection.

First part included calculating the average time it takes to compute a route. We chose 10 different kind of routes in terms of length and road types from Helsinki metropolitan area. Testing was done with Python script that called the network service API:s and the local services on the server and recorded the time that the 10 route calls took on each service. The measured time included only the calls to the services and not the initialization of the call data and parsing of the results. This process was done 10 times so every service was called 100 times and average route call time was calculated from these 100 results. There was a clear difference between the local- and network services. Local services were at least 10 times faster than the network services even with really fast internet connection. OSRM had the lowest calculation time with 6 milliseconds and MapQuest highest with 451 milliseconds.

Second part included testing the route times that services provided with the routes against real driving times in these routes. We chose 5 routes that employees of Vediafi drive regularly and are familiar with. This makes it possible to gather the data naturally without the need to make dedicated test runs and makes sure that the driving times are optimal. Because the sample size was pretty small, we also chose 8 routes that represent likely taxi rides. These routes go between the Helsinki airport, hotels, industrial areas, entertainment centers and the western harbour of Helsinki.

After 15 test runs on the routes it became clear that the local services had much bigger error rate in their estimate compared to the actual driving

time than online services. Google Maps had the lowest error rate with 12% and Routino the highest with 42%.

Visual analysis on the routes that the employees drove regularly was also done by drawing the routes provided by all the services to a single map. Most of the time all of the services provided the route that the experienced driver chose. Few of the routes took a longer route with faster roads and it was the preference of the driver to take the shorter route which was marginally slower.

Service	Online	Price	Route calculation time	Route time error
Routino	No	Free	27ms	42%
OSRM	No	Free	6ms	32%
MapQuest	Yes	15000 free requests per month, next 15000 requests 99 USD	451ms	18%
Google Maps	Yes	2500 free requests per day, next 1000 requests 0.5 USD	275ms	12%

Table 4.1: Comparison of the routing services

[52] [26] [53] [46]

4.4 The ride finding algorithm in Vedia Taxi

The goal of the algorithm is to find suitable rides for the users to join. Ride is suitable when joining it saves money for the user and the time needed to complete the ride does not rise too much compared to the direct route when riding alone in a taxi for any of the passengers. The algorithm takes the user's origin and desired destination as input and gives the suitable rides found from the database as output. The output includes the new combined route for every ride as it would be after the user joins the ride, so user can compare the rides properly.

4.4.1 Versatility

In the current state, the ride finding algorithm assumes that every passenger joins the ride from the same location. User can spesify what distance they are willing to walk to get to the starting location. This restriction was

Route	OSRM	Routino	MapQuest	Google Maps	Actual time(s)
Valimotie 13 - Hotelli Haaga	4.5 min	4.4 min	7.2 min	9.4 min	8.5 min
Hotelli Haaga - Messukeskus Helsinki	4.9 min	5.0 min	7.1 min	7.8 min	5.5 min
Messukeskus - Helsinki Airport	13.9 min	12.9 min	15.1 min	16.5 min	16.1 min
Helsinki Airport - Holiday Inn Helsinki - Vantaa	4.0 min	3.9 min	7.4 min	5.4 min	4.8 min
Technopolis - Flamingo	4.5 min	4.2 min	7.0 min	6.3 min	5.7 min
Flamingo - Omenahotelli Lönnrotinkatu	18.9 min	17.1 min	20.7 min	32.2 min	28.8 min
Omenahotelli Lönnrotinkatu - Western harbour of Helsinki	4.4 min	4.4 min	4.8 min	9.3 min	4.7 min
Western harbour of Helsinki - Valimotie 13	12.8 min	11.3 min	15.4 min	24.7 min	21.0 min

Table 4.2: Likely taxi trip routes. Actual times were acquired by driving the routes with a car.

Route	OSRM	Routino	MapQuest	Google Maps	Actual time(s)
Valimotie 13 - Tuurinpohja 4	13.1 min	11.1 min	15.4 min	16.7 min	16.7 min
Valimotie 13 - Tiilenpolttajankatu 24	75.8 min	61.9 min	67.5 min	72.0 min	65.5, 70.2 min
Valimotie 13 - Isonmastontie 13	21.1 min	20.4 min	24.4 min	31.7 min	32.0, 34.0 min
Tiilenpolttajankatu 24 - Piisitie 1	8.6 min	5.7 min	9.3 min	10.7 min	9.4 min
Tietotie 2 - Veturitie 20	12.4 min	11.5 min	15.8 min	17.1 min	16.4 min

Table 4.3: Routes regularly driven by Vediafi employees

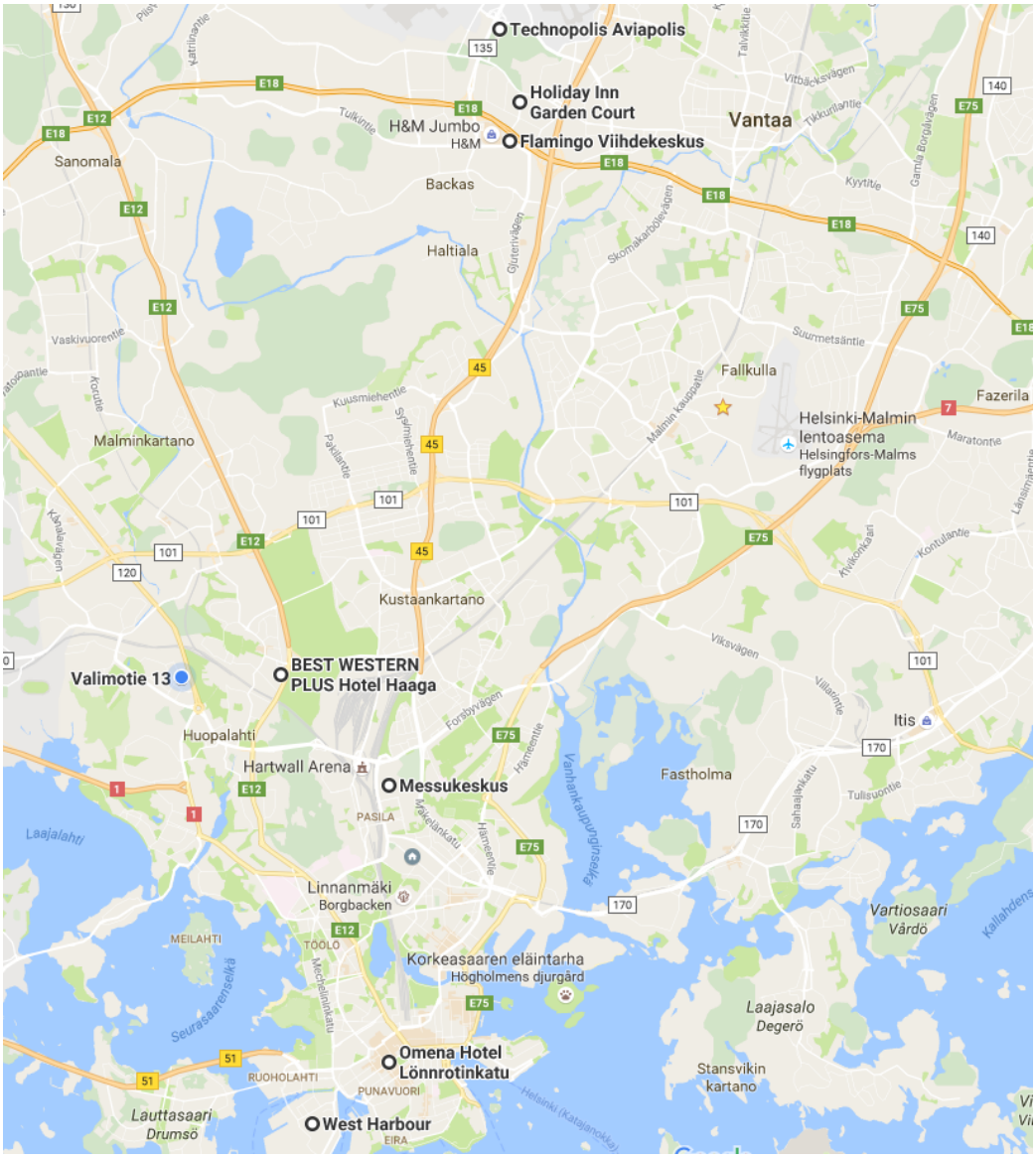

Figure 4.1: Likely Taxi trip locations (Helsinki Airport outside of the picture)

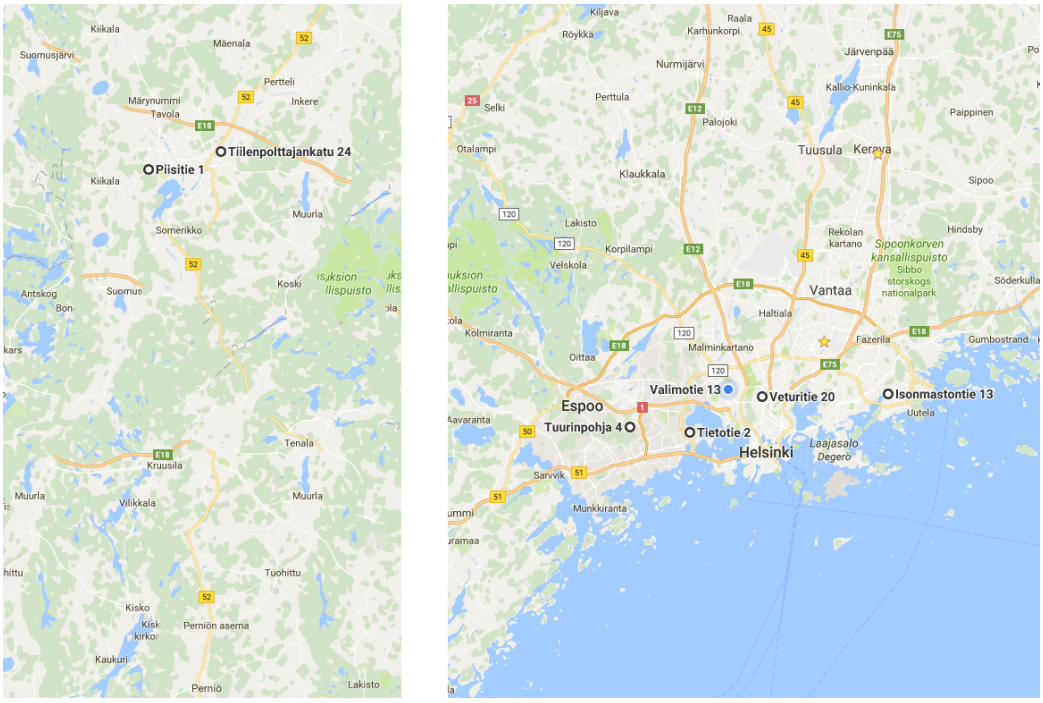

Figure 4.2: Vediafi employee route locations

implemented because it greatly simplifies the algorithm and ensures that all the passengers are present before the taxi leaves the starting location.

The purpose of this work is to extend the algorithm to be more versatile, so that you can join the ride from the route too. This does not mean, however, that passengers could join during the actual ride. This restriction ensures that everyone knows the route and their share of the prices before the ride begins. Extra passengers joining during the ride would also change the times of arrival for current passengers.

This change also means that when a passenger joins the ride they might also become the new first passenger and thus must order the taxi. This needs to be communicated clearly to all passengers when a change like this happens. In the current implementation the passenger who leaves the taxi last can change too when a new passenger joins or the current owner modifies the route. All the passengers are notified about the change in this scenario too via push notifications.

4.4.2 Possible general solution

Basic idea for the solution to the new algorithm contains two stages. We have the user's origin and desired destination as input and database that we can query for rides.

First we go through all the rides and check if any of the rides have a route that goes near the user's origin. This check can be done by creating a buffer around the points in the route and checking if the user's origin is inside this buffer. If there are such rides, we form new routes that include the user's origin and destination. If these routes are suitable, as defined in section 4.4, we add the new routes and rides to the output list.

In second phase we form a route with the user's origin and destination. Then we go through all the rides again and check if any of them have starting points near the newly formed route. If they do, we combine the routes as in first phase and check if the new routes are suitable. If they are, we add the new routes and the rides to the output list, if they are not already there.

4.4.3 Restrictions and possible optimizations to the general solution

While the general solution to the algorithm could work as is, there are several possible optimizations and restrictions that we need to take in to account when starting to implement it.

When we go through the rides, we should try to limit the amount of rides we query from the database, as some of the rides could be in very different geographical location as the user origin. One of the ways to do this could be to calculate bounding boxes for the routes as rides are created and then compare the bounding boxes when doing the search. This or some other geographical restriction could reduce the rides that we must go through significantly.

As we search through the rides we need to carefully define what is suitable value of route being near to a point. The value cannot be too large to avoid getting too many hits and not too small to not miss valid rides. The fact that in a road network the real distance between the route and a point can be much longer than the direct distance on the earth's surface complicates things too. We could calculate the real distance from each point in the route to the specific point, but that could be too expensive as route calculation is much more time consuming than calculating a straight distance.

When we combine routes in the algorithm, we need to correctly determine the order of waypoints in the new route. When we insert the user's origin and destination, we need to know in which leg of the route they are inserted.

Google Maps and MapQuest offer possibility in their routing service to automatically determine the best order for waypoints. [52] [26] With the local services we would need to determine this manually.

All the rides also have a starting time which means that we have to be sure that when we are combining the rides everyone needs to be able to be picked from their location by the starting time they have spesified when they created or searched for the ride. For example, if a new route in the second phase in the algorithm is found and the user becomes the new first passenger in the route, we must check that the user has enough time to get to the next location, before passenger in that location needs to be picked up.

We must also determine a suitable minimum for the amount of money that the combining of rides saves for each of the passengers. As every passenger pays a provision to Vediafi and the route that the passengers share might be really short the passengers might end up paying more from the ride than riding alone. There is also the case where a new passenger joins for a really short part of the ride and raises the kilometer based fee of the fare to a new fare class and the passengers that were on the ride before end up paying more because of the new passenger joining.

Chapter 5

Extended algorithm

In this chapter we go through the old route finding algorithm and its evolution to the new version. We go through the final version of the algorithm and present the problems faced and the changes from the possible solution presented in chapter 4.4. There is also a new use case with the extended algorithm and summary of new problems that the extended algorithm may cause.

5.1 Choosing the routing service for the algorithm

We compared four different routing services in chapter 4.3. We ultimately chose Google Maps when implementing the algorithm. This was mostly because the approximated route time was most accurate with Google Maps and it is crucial for the algorithm to determine if passengers can be picked up in time from the route. The route time error in MapQuest and Google Maps was really close, but Google Maps also had faster query times between these two. They both were free up to a certain limit. The local services had much better query times, but the route time errors were simply too big.

5.2 Extended algorithm implementation

5.2.1 Implementation overview

When we started to develop the new algorithm, we took the old algorithm as a starting point. We only needed to remove the start point check from

the old algorithm, because other parts would be needed in the new algorithm too.

```
func find_eligible_rides(all_rides, user_start_point, user_end_point)
    eligible_rides = []
    second_phase_rides = []

    for (ride in all_rides)
        if (ride.start_point too far away from user_start_point)
            continue

        if (ride does not have enough seats left)
            continue

        if (user_end_point nearby ride.route)
            if (combined route is fast enough)
                eligible_rides.add(ride)
        else
            second_phase_rides.add(ride)

    for (ride in second_phase_rides)
        user_route = get_direct_user_route(user_start_point, user_end_point)

        if (ride.end_point nearby user_route)
            if (combined route is fast enough)
                eligible_rides.add(ride)

    return eligible_rides
```

Table 5.1: The old algorithm

We started adding the restrictions and checks listed in chapter 4.4 and we quickly realized that many of these checks have to be performed twice if we use the two phase approach to the problem. We noticed that we could merge the check for starting and ending points in the same loop, eliminating the need for second phase. This way the algorithm became simpler and no operation was needlessly done twice.

The body of the new algorithm is basically one loop that goes through all the rides and a setup phase. In the setup phase we initialize the user route, so we need to fetch it only once. The big thing when thinking about the implementation was minimizing the amount of route queries we have

to make, because they are so slow compared to everything else. Each loop checks the requirements one by one for a single ride and if all the conditions are met, the ride is added to the result list.

```
func find_eligible_rides(all_rides, user_start_point, user_end_point)
    eligible_rides = []
    user_route = get_direct_user_route(user_start_point, user_end_point)

    for (ride in all_rides)
        if (user_route.bounding_box does not intersect
            ride.route.bounding_box)
            continue

        if (neither route has the other routes end point nearby them)
            continue

        if (neither route has the other routes start point nearby them)
            continue

        combined_route = get_combined_route()
        if (combined_route cannot be formed because of time restraints)
            continue

        if (combined_route is not fast enough)
            continue

        if (combined_route does not have enough seats left)
            continue

        if (combined_route is not cheap enough)
            continue

        eligible_rides.add(ride)

    return eligible_rides
```

Table 5.2: The new algorithm

Because the individual executions of the loop body are not related to each other we considered doing them in parallel. For our algorithm this probably

would not have been a problem, but Google Maps has limitation on how many requests per second you can make to the server, currently it is 50. This could pose a problem when there are more users and more concurrent searches. Even one search could sometimes break this limit during rush hours. Even if we cannot do the executions in parallel, most of the rides are quickly discarded with the bounding box and start point checks before the more time consuming route fetches.

5.2.2 Detailed explanation of the loop body

Here we go through the statements in the main loop body one by one.

```
if (user_route.bounding_box does not intersect ride.route.bounding_box)
    continue
```

First we do a bounding box check to ensure that the bounding box of the users route overlaps with the bounding box of the rides route. This is done to quickly discard all the rides that are too far away to be meaningful candidates for route combining. Sometimes this check discards rides that could have been combined, if the bounding boxes are really close and the close parts are not the end of the routes. This flaw could possibly be avoided by expanding the bounding boxes beyond the route borders, but for simplicity this check was left as is, with the possibility of some false negatives.

```
if (neither route has the other routes end point nearby them)
    continue

if (neither route has the other routes start point nearby them)
    continue
```

Even if the bounding boxes overlap, we have to check if the routes have meaningful possibilities to form combined route. We do this by inspecting the ending and starting points. If we have route A and route B as the routes, we must have a situation where either starting point from route A is along route B or starting point from route B is along route A. Similarly we check that A or B has their ending point along the other route. If both of these checks return a positive result, we know that we can form a route with one of the following waypoint orders:

- $A.start \rightarrow B.start \rightarrow A.end \rightarrow B.end$
- $B.start \rightarrow A.start \rightarrow A.end \rightarrow B.end$

- $A.start \rightarrow B.start \rightarrow B.end \rightarrow A.end$
- $B.start \rightarrow A.start \rightarrow B.end \rightarrow A.end$

To determine if a point is along a route we calculate distances between the point and all the points in the route. The distance used is a simple great-circle distance along the earth calculated with Haversine formula [54]. The real distance of two points in a road network is different than the great-circle distance, but getting the real distance to each point in the route would be far too costly. This can result false positives if there is a river or other obstacle between the points which makes the real distance much longer than the great-circle distance. The phase where we make the combined route will discard these false positives.

If even one point in the route is close enough to the desired point we return a positive result. We must also determine what distance is close enough to the route and we use a value that is proportional to the routes length. Current value is 20% of the routes length. We thought that longer routes could afford longer detours to fetch new passengers, because the savings are bigger on longer routes.

```
combined_route = get_combined_route()
```

Next we form the combined route. Before we can form it we must determine the correct order for the waypoints. We use simple great-circle distances again to save the route calls. The difficulty here is that in order to calculate the ideal starting point we would need to know the ideal ending point and vice versa. We could not get around this problem so we chose to use the end point of the route from the ride as ideal end location when determining optimal start location. When we have the ideal start location we check again what would be the ideal end location. The ideal end and start location are as far away from each other as possible. This could present some unoptimal orders sometime, but users are offered the possibility to reorder stops if the route seems unoptimal. The rest of the waypoints that are in the route are optimized by Google Maps when we request the route.

```
if (combined_route cannot be formed because of time restraints)
    continue
```

When we have the combined route we also have the times that each leg of the route will take. We must then check that the new passenger can be picked up before their desired leaving time. The new passenger might also be the starting passenger in which case we must check that the new passenger has

time to pick up the old passenger along the route. We can do this easily by going through the route and checking the times for each leg and comparing them to the desired leaving times of each passenger.

```
if (combined_route is not fast enough)
    continue
```

We must also check that the new route is not unexpectedly slow. Here we have two values that define how much longer can the new route be compared to the old one. The route can be 50% longer, if the new route is not extended from the beginning or the end and 150% longer if the route is extended. These values are relatively high, because we don't want to discard too many rides in this phase when the most expensive part, the route fetch, is already done. This way we can offer more choices to the users to choose from when they search for the rides.

```
if (combined_route does not have enough seats left)
    continue
```

Unlike the old algorithm, we cannot check if there are enough seats left based on just the amount of passengers in the start location, because passengers join and leave during the route. We have to check that there is enough space during the whole route. To ensure that we go through the waypoints one by one and keep track on the number of passengers and if that number exceeds the maximum amount at any point.

```
if (combined_route is not cheap enough)
    continue
```

Finally we must check if joining the ride costs more than riding alone. This can happen when you are riding alone for a long stretch and share only a small part of the ride, because provisions paid to Vediafi are proportional to the price of the whole ride. So we calculate the amount passenger would pay in the shared ride and the amount they would pay when using their own route alone and compare them. In chapter 4.4 there was a mention that ride could also be more expensive if the ride would be upgraded to higher kilometer based fee when new passenger joins. After some research it turned out that the kilometer based fee changes dynamically during the ride and is not based on the highest amount of passengers that are in the ride at the same time. [55]

5.3 General use case for the application with extended algorithm

In this example we have three persons, Alice, Bob and Carol. All of them have used Vedia Taxi before.

Alice is leaving from her home to see a rock concert. She decides to create a ride in Vedia taxi to save some money. Bob is going to the same concert, but lives further away from the city. He finds the ride Alice created and joins it, becoming the new first passenger so he must order the taxi. Carol is already in the city where the concert is and needs a short ride through the city. She finds the ride too and joins it. Her destination is bit further than the concert hall, so she becomes the new owner of the ride and has to pay for the taxi.

Bob orders the taxi and first picks up Alice and then Carol when the taxi is in the city. Bob and Alice pay their share of the ride to Vediafi and leave the taxi near the concert hall and Carol continues to her destination and pays for the taxi. All of them mark the ride to be succesful and give reviews to the other passengers.

5.4 Summary of potential problems with the extended algorithm

Here is a summary of new problems that appear after the application is used with the extended algorithm. This does not include problems presented in chapter 3.2.

- First passenger is changed and he does not get the push notification that he should be the one to order the taxi and pick up other passengers. This situation does not have a good solution if the passenger is not paying attention to the application.
- Passengers leaves the ride and after the route is recalculated there is a gap where no passengers are in the taxi. In this case the ride has to be canceled.
- Passenger that was supposed to be picked in the middle of the ride does not show up and it causes a gap in the route where no passengers are in the ride. The taxi can be directed to the next passenger or the rest of the ride can be canceled. The passengers may ask Vediafi for

refunds on the cancelled legs and the extra trip that taxi rides without passengers.

Chapter 6

Evaluation

6.1 Testing setup

The ideal way to test the algorithm would have been with real users and real rides, but Vedia Taxi has not gained enough popularity yet and there is not enough users and rides to get data on how well the algorithm is performing.

Instead, we formulated test cases to validate that the different aspects of the new algorithm work. Test cases are searches with the algorithm from database of four rides. The ride routes and searches represent realistic situations that real users would encounter. Test cases were implemented in the same server code base as the algorithm.

These test cases do not prove that the algorithm works on every situation. They showcase that rides can now be formed in four new ways and that uneligible rides are properly discarded during different parts of the algorithm.

There are four different types of new routes when passenger joins that we tested for:

1. Passenger joins and the original route is extended from the start
2. Passenger joins and the original route is extended from the end
3. Passenger joins and the original route is extended from both start and end
4. Passenger joins and the original route is augmented by stops that are within the original start and end points

These are the test routes in the database:

1. Helsinki Airport - Helsinki Railway Station

2. Linnanmäki - Salmisaaren Liikuntakeskus
3. Korkeasaari zoo - Reposalmenpolku
4. Olympic Stadium - Brewdog Helsinki

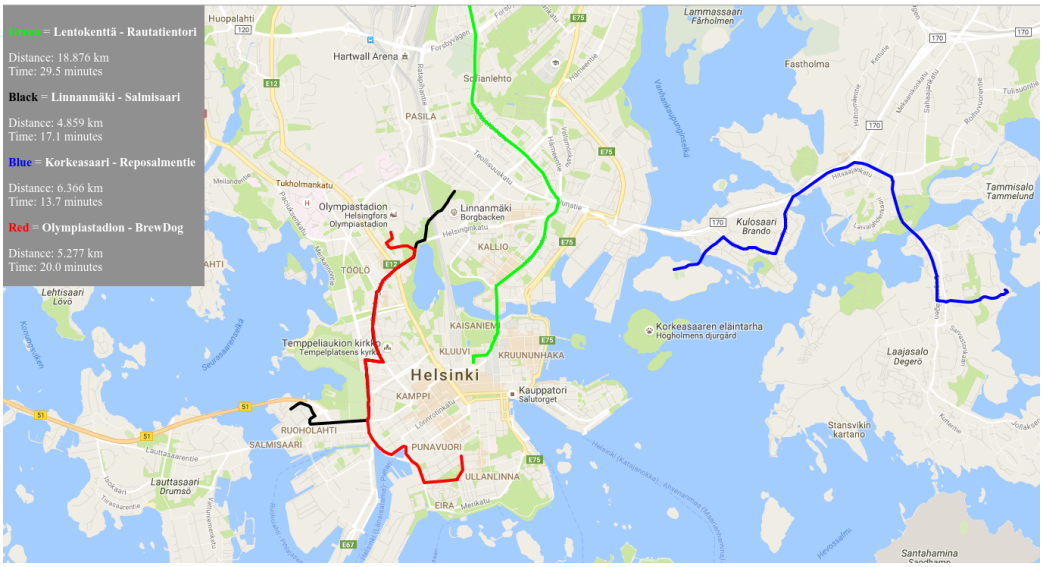

Figure 6.1: Test routes (Helsinki Airport outside of the picture)

And these are the test searches that queried eligible rides from the database:

1. Helsinki Airport - Flamingo
2. Flamingo - Hotel Torni
3. Messukeskus Helsinki - Omena Hotel Lönnrotinkatu
4. Hartwall Arena - Blue Peter Restaurant and Conference Center
5. Hilton Helsinki - Tavastia
6. Jorvi Hospital - Töölö Hospital

6.2 Test searches

Test search 1 demonstrated the only search that the old algorithm could handle, both the original passenger and the joining passenger leave from the

same location. The algorithm yielded the ride 1 as the result for the search and the new route was of type 4, because the route was not extended from either end.

Test search 2 again returned ride 1 as the result, but this time the type of the new route was 2, as the new passenger leaves last.

With test search 3 we got two results. Ride 2 was the first result with new route of type 1, the new passenger is the new first passenger and extends the route from the beginning, but leaves before the end. Ride 4 was similar kind of result, but the route was just a bit different. Ride 1 could have been be eligible as well, but in this case the bounding box check discarded it.

Test search 4 yielded one result, ride 2. This time the new route was of type 3, the old route was completely inside the new route and the new passenger is both the new first and last passenger.

Test search 5 returned ride 4 as the results and the new route was of type 1

Finally the test search 6 returned no rides as a result. If just the eligibility of the route would be a factor, ride 4 would have sufficed, but the provisions caused by taking most of the trip alone were bigger than the money saved by sharing the taxi.

With these test searches we had all the route types tested. None of the searches yielded ride 3 as a result and this was intended.

The start time and seat limitations were tested with same searches, but with different initial person counts and start times. The searches yielded no results when there were too many passengers on any position on the route or there was no time to reach the next stop.

6.2.1 Detailed explanation of test search 3

Here is a detailed breakdown why rides were discarded and accepted on test search 3. In figure 6.2 and figure 6.3 are the rides and search query before and after the combining is complete.

Ride 1 is discarded, because the bounding box check fails. The bounding boxes are actually really close, and a combined route might be possible to form if the ride was not discarded by the bounding box check. Ride 3 is also discarded because of the bounding boxes.

Ride 2 clears the bounding box check. The start point of ride 2 is on the searched route and the end point of the searched route is on the route of ride 2, so we can continue to combine the routes. There is no time restraints and the combined route time is within the limit. There is enough seats left and the price is cheaper with the new route than riding single. Ride 4 is accepted with similar process.

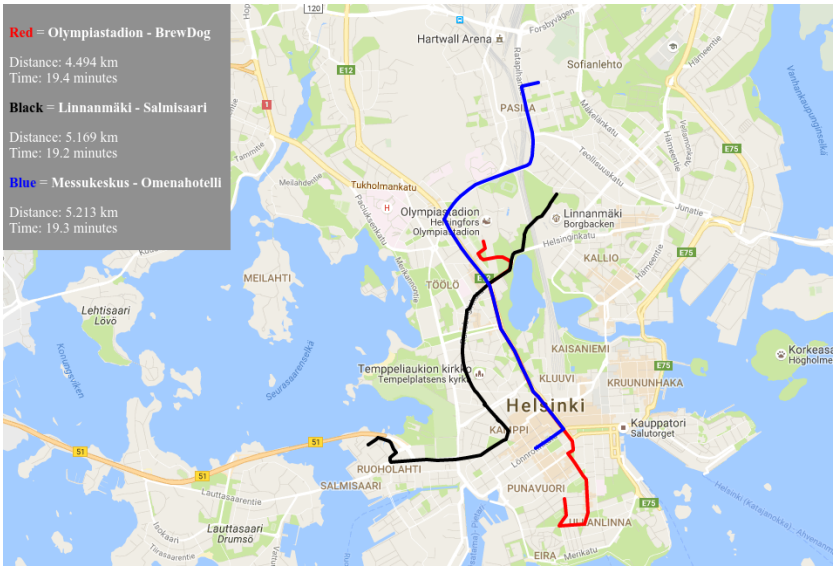

Figure 6.2: Test search 3, blue line is the searched route, red and black are the routes that exist in database

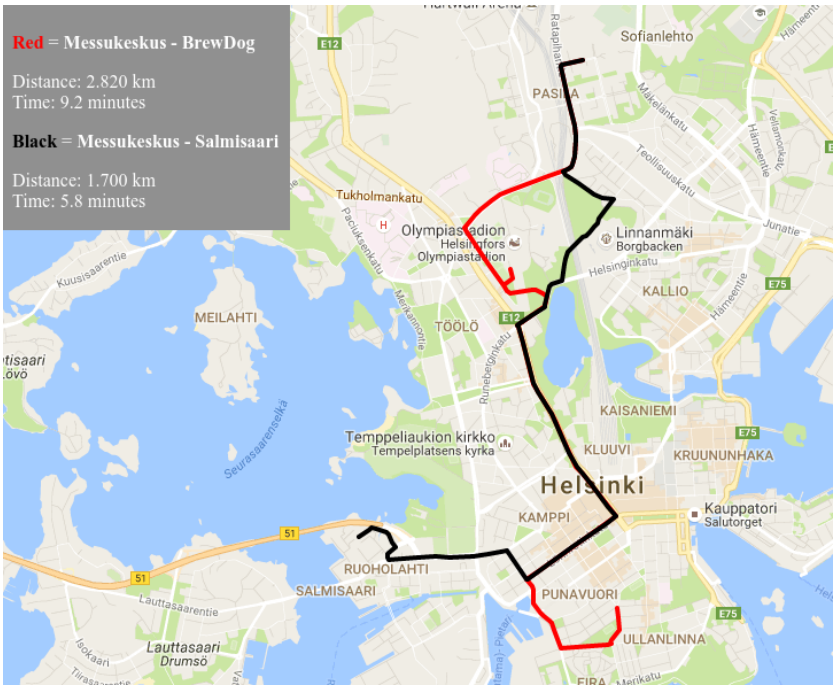

Figure 6.3: Resulting combined routes from test search 3

6.3 Performance testing

Performance of the algorithm was tested with the same test searches, but this time we measured the time it took to complete different parts of the search. We measured time that it took for the whole algorithm to complete, the setup phase time before the main loop and time that was need to complete each individual loop. If route fetch was made we recorded the amount of time it took from the whole operation.Tests were run 10 times to ensure the reliability of the results.

As we expected, the route fetches consumed majority of the time needed by the algorithm, between 92% and 98%. If no route fetching was needed, the time to discard the ride with bounding box and end point analysis was 0-1 milliseconds. If the route was fetched, the decision about the eligibility of the ride was made in 15-25 milliseconds if we do not include the time that was needed to fetch the route. The time that was needed to fetch the route was analysed in chapter 4.3. The time to complete the setup phase was practically completely consumed by the initial fetch of the user route. The whole algorithm took between 200ms and 2200ms to complete depending on the amount of route fetches we had to do and how many rides we could not quickly discard.

Chapter 7

Discussion

From the services presented in chapter 2.2, the new Vedia Taxi is most similar to KutsuPlus. In both services people share a ride and people can join and leave the ride in a similar fashion. The key difference is that Vedia Taxi only offers people a way to share their ride, but does not operate the taxi service itself, like KutsuPlus did.

Airport Taxi also operates by sharing the ride and reducing costs, but it is a very spesific service and requires you to be leaving or going to the airport. Uber could be included in our service, as it is kind of a taxi service. The difficulty with Uber is that it has so called surge pricing that fluctuates the prices unpredictably.

If the Vedia Taxi service gains more popularity, the issue of scalability could become a problem. The searches could take longer, as there would be more and more rides in smaller areas that could not be discarded with the fast methods such as the bounding box check. This problem could be solved in a few different ways. We could return only the first N results or run the search only for a certain time so it would not take too long. The rides could be evaluated concurrently, but the problem is the limit of concurrent searches we can make to the routing service. This could be solved by asking the routing service for more concurrent searches or trying to tune the local services so that they would return better time estimates.

Vedia Taxi uses third party services for routing and payments and if any of these services has an outage or they quit offering the service, Vedia Taxi can't function. Having a backup service is feasible for the routing service, but the payment service is a bit more difficult as the users credit card information is stored in their server. The routing service provider was already updated from MapQuest to Google Maps, because the old algorithm used MapQuest and the comparison showed that Google Maps is better. The migration did not take a long time, because the feature sets of these two services were really

similar. Migrating to a new payment solution could take more time, as there are many interfaces that need to be changed and the user data needs to be migrated from the old service to the new one.

Chapter 8

Conclusions

This chapter concludes the thesis by providing a summary of the results and answers to the research questions. This chapter also has discussion about the need for further research for the extended algorithm.

8.1 Summary of results

The purpose of this thesis was to extend the algorithm that Vedia Taxi uses to find and combine routes. The original algorithm could only find routes which had roughly the same start location and users could not join from along the route. With the new extended algorithm routes can also be formed when the start locations are different. This also makes it possible to join along the route. The features of the new algorithm were tested with realistic test searches that validated that the features work at least in these spesific test cases. They, however, do not prove that the algorithm works on every situation.

Comparison of different directions services was made and there were four candidates: Google Maps, MapQuest, Routino and OSRM. Google Maps was selected as the directions provider, because it had the best results when measuring the accuracy of the time estimates. The accuracy of the time estimates was tested with test drives. The routes from different providers had some differences, but all of them were adequate. Routino and OSRM had the best route calculation times, but their route time estimates were over 30% off, so they could not be used.

The performance of the algorithm was measured and the algorithm in itself didn't take more than 25 milliseconds per ride to evaluate if a ride is eligible. The calls to Google Maps directions service took the majority of the time that the algorithm needed. The average time that one Google Maps

directions call took was measured to be 275ms. Before a call to directions service is made the algorithm makes several geographical checks to ensure that the route is likely eligible. These checks did not take more than 1ms to execute. In conclusion, we can say that the algorithm perfoms fast enough.

One of the research questions was about how we could incorporate disruption data to the routes. This proved to be a very difficult task, as traffic disruption data was not openly available, but usually the directions providers automatically used their own data to enhance the directions. Google Maps uses historical and real time traffic data as well as traffic incident data to enchance their routes and time estimates. By using Google Maps as our directions provider we have this information also on our routes. [41]

8.2 Future research

There are several things that the future research on this subject could focus on.

The bounding box check that the algorithm performs is very strict and could be improved. Unlike the start and end point analysis on the route, it does not have any flexibility and so it can discard some rides that would be possible to combine, when the bounding boxes are really close to each other.

As the most time in the algorithm is used in the directions service calls, trying to minimize that time would be a huge step for the algorithm. Both of the local routing providers have a huge selection of parameters and settings that influence the resulting route and the time estimate. Trying to find optimal parameters was not in the scope of this thesis. With the right parameters and settings the local services could provide more accurate time estimates and could be considered as an alternative to the web based services. This would lower the time the algorithm takes to complete significantly. It would also enable the possibility to multithread the search process, as the strict limitation of concurrent searches would disappear.

Finally, if the service gains popularity, some work should be done on gathering real information about the performance of the algorithm and the experiences from the users.

Bibliography

[1] Matthew Feeney and Rideshare companies Uber. Is ridesharing safe? *Cato Policy Analysis*, 767:2, 2015.

[2] Joonas Stenroth. Joukkoliikenteen matkustajalaskenta helsingin seudulla. Master's thesis, Tampere University of Technology, 2015.

[3] ShÄre-a-taxi, March 2016.

[4] Harald Heinrichs. Sharing economy: a potential new pathway to sustainability. *GAIA-Ecological Perspectives for Science and Society*, 22(4):228–231, 2013.

[5] Boyd Cohen and Jan Kietzmann. Ride on! mobility business models for the sharing economy. *Organization & Environment*, 27(3):279–296, 2014.

[6] Aalto marketplace, March 2016.

[7] Juho Hamari, Mimmi Sjöklint, and Antti Ukkonen. The sharing economy: Why people participate in collaborative consumption. *Journal of the Association for Information Science and Technology*, 2015.

[8] The state of the sharing economy, May 2013. Report by The People Who Share.

[9] Ron Buliung, Kalina Soltys, Catherine Habel, and Ryan Lanyon. Driving factors behind successful carpool formation and use. *Transportation Research Record: Journal of the Transportation Research Board*, (2118):31–38, 2009.

[10] Nelson D Chan and Susan A Shaheen. Ridesharing in north america: Past, present, and future. *Transport Reviews*, 32(1):93–112, 2012.

[11] Jianling Li, Patrick Embry, Stephen Mattingly, Kaveh Sadabadi, Isaradatta Rasmidatta, and Mark Burris. Who chooses to carpool and why?: examination of texas carpoolers. *Transportation Research Record: Journal of the Transportation Research Board*, (2021):110–117, 2007.

[12] Niels Agatz, Alan Erera, Martin Savelsbergh, and Xing Wang. Optimization for dynamic ride-sharing: A review. *European Journal of Operational Research*, 223(2):295–303, 2012.

[13] Yeqian Lin, Wenquan Li, Feng Qiu, and He Xu. Research on optimization of vehicle routing problem for ride-sharing taxi. *Procedia-Social and Behavioral Sciences*, 43:494–502, 2012.

[14] Shuo Ma, Yu Zheng, and Ouri Wolfson. T-share: A large-scale dynamic taxi ridesharing service. In *Data Engineering (ICDE), 2013 IEEE 29th International Conference on*, pages 410–421. IEEE, 2013.

[15] Corinne Mulley and John D Nelson. Flexible transport services: A new market opportunity for public transport. *Research in Transportation Economics*, 25(1):39–45, 2009.

[16] Suomen taksiliitto, March 2016. Fetched 25.03.2016.

[17] Airport taxi, March 2016. Fetched 25.03.2016.

[18] Uber, March 2016. Fetched 25.03.2016.

[19] Faktalehti 22/2015, December 2015.

[20] Noona Bäckgren. Poliisi otti uber-kyydit hampaisiinsa helsingissa ¿ kuskeille useita kymmeniä sakkoja. *Helsingin Sanomat*, December 2015.

[21] Sanna Vikman. Kutsuplus hyllytettiin vuodeksi ¿ hsl etsii yrittäjää tekemään palvelusta kannattavan. *Yle*, November 2015.

[22] Centos, April 2016.

[23] Django, April 2016.

[24] Postgresql, April 2016.

[25] Google cloud messaging, April 2016.

[26] Mapquest directions api, April 2016.

[27] Android, April 2016.

[28] Android studio, April 2016.

[29] Google sign-in, April 2016.

[30] Facebook login, April 2016.

[31] Android market share in finland, October 2014.

[32] Android market share globally, April 2015.

[33] Stripe, April 2016.

[34] Liping Fu, D Sun, and Laurence R Rilett. Heuristic shortest path algorithms for transportation applications: state of the art. *Computers & Operations Research*, 33(11):3324–3343, 2006.

[35] Jørgen Bang-Jensen and Gregory Z Gutin. *Digraphs: theory, algorithms and applications*. Springer Science & Business Media, 2008.

[36] George H Polychronopoulos. Stochastic shortest path problems with recourse. *Networks*, 27(2):133–143, 1996.

[37] Global forecast system, May 2016.

[38] The european centre for medium-range weather forecasts, May 2016.

[39] Guillaume Leduc. Road traffic data: Collection methods and applications. *Working Papers on Energy, Transport and Climate Change*, 1(55), 2008.

[40] Juan C Herrera, Daniel B Work, Ryan Herring, Xuegang Jeff Ban, Quinn Jacobson, and Alexandre M Bayen. Evaluation of traffic data obtained via gps-enabled mobile phones: The mobile century field experiment. *Transportation Research Part C: Emerging Technologies*, 18(4):568–583, 2010.

[41] Tim Stenovec. Google has gotten incredibly good at predicting traffic ¿ here's how. *Tech Insider*, November 2015.

[42] Roark Weil, J Wootton, and A Garcia-Ortiz. Traffic incident detection: Sensors and algorithms. *Mathematical and computer modelling*, 27(9):257–291, 1998.

[43] Thiago H Silva, Pedro OS Vaz de Melo, Aline Carneiro Viana, Jussara M Almeida, Juliana Salles, and Antonio AF Loureiro. Traffic condition is more than colored lines on a map: characterization of waze alerts. In *Social Informatics*, pages 309–318. Springer, 2013.

[44] Mordechai Haklay and Patrick Weber. Openstreetmap: User-generated street maps. *Pervasive Computing, IEEE*, 7(4):12–18, 2008.

[45] Pascal Neis, Dennis Zielstra, and Alexander Zipf. The street network evolution of crowdsourced maps: Openstreetmap in germany 2007–2011. *Future Internet*, 4(1):1–21, 2011.

[46] Openstreetmap wiki - stats, May 2016.

[47] Mapping usage statistics, May 2016.

[48] Maps market share in the alexa top 1m, May 2016.

[49] Google map maker, May 2016.

[50] Błażej Ciepłuch, Ricky Jacob, Peter Mooney, and Adam Winstanley. Comparison of the accuracy of openstreetmap for ireland with google maps and bing maps. In *Proceedings of the Ninth International Symposium on Spatial Accuracy Assessment in Natural Resuorces and Enviromental Sciences 20-23rd July 2010*, page 337. University of Leicester, 2010.

[51] Google maps pricing and plans, 2016.

[52] Google maps directions, May 2016.

[53] Routino, May 2016.

[54] Glen Van Brummelen. *Heavenly mathematics: The forgotten art of spherical trigonometry*. Princeton University Press, 2013.

[55] Private communication, July 2016. Information acquired 26.07.2016.